

Quasi-Newton Methods

Laboratory work 3

Leontev Taras, Shpileva Anastasiya, Maria
Okorochkova

Gauss-Newton, Powell Dog Leg, BFGS, L-BFGS



Department of Computer Technologies
University ITMO
Teacher Kazankov VK. 13.05.2023

1 Gauss-Newton and Powell Dog Leg

1.1 Gauss-Newton

Мы хотим научиться решать задачи нелинейной регрессии.

$$f(x) = \sum_{i=1}^m r_i(x)^2$$

тут r - квадраты расстояние до целевой функции (кривой).

$$r_i = y_i - p(t_i)$$

Эта задача решается методом Ньютона.

$$x^{k+1} = x^k - (\nabla^2 f(x^k))^{-1} \nabla f(x^k)$$

Но мы столкнемся с тем, что вычисления Гессиана не дешево а также, что наш спуск сильно зависит от выбранной нами начальной точки. Давайте попробуем выразить Гессиан и Градиент через матрицу Якоби.

$$f(x) = \frac{1}{2} \sum_{i=1}^m r_i(x)^2$$

$$\frac{\partial f}{\partial x_j} = \frac{1}{2} \cdot 2 \sum_{i=1}^m \frac{\partial r_i}{\partial x_j} r_i$$

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial r_0}{\partial x_0} & \cdots & \frac{\partial r_0}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_0} & \cdots & \frac{\partial r_m}{\partial x_n} \end{pmatrix}^T \cdot \begin{pmatrix} r_0 \\ \vdots \\ r_m \end{pmatrix}$$

$$\nabla f(x) = J_r^T r, \quad J_r = \left[\frac{\partial r_i}{\partial x_j} \right] \quad i = 1..m, \quad j = 1..n$$

Чтобы получить Гессиан надо продифференцировать еще раз.

$$\frac{\partial f}{\partial x_j} = \sum_{i=1}^m \frac{\partial r_i}{\partial x_j} \cdot r_i$$

$$\frac{\partial f}{\partial x_j \partial x_k} = \sum_{i=1}^m \left(\frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k} + r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k} \right)$$

Откуда получаем

$$\nabla^2 f = J_r^T J_r + Q$$

$$Q = \sum_{i=1}^m r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k}$$

В методе Гаусса-Ньютона мы упускаем Q

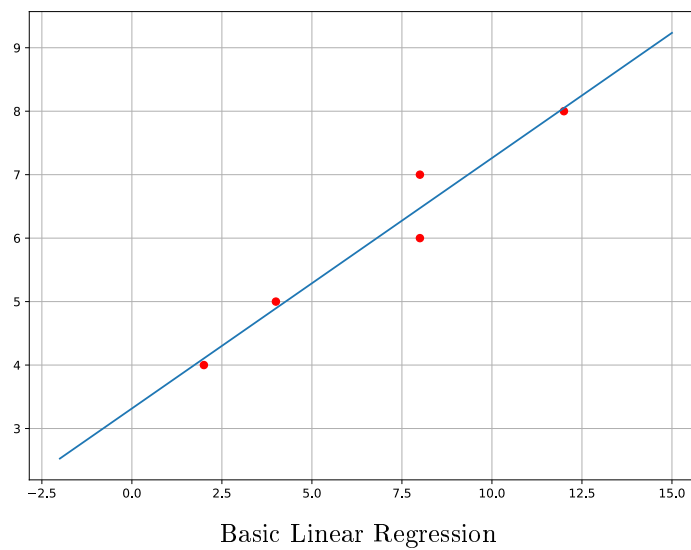
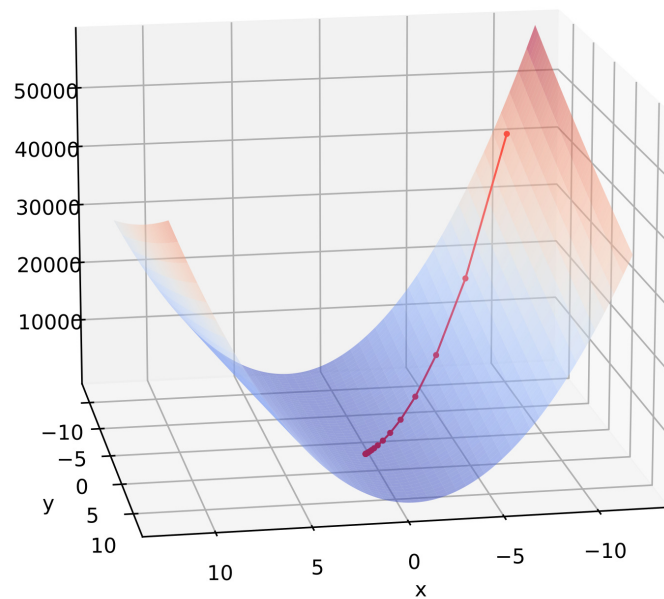
$$\nabla^2 f \approx J_r^T J_r$$

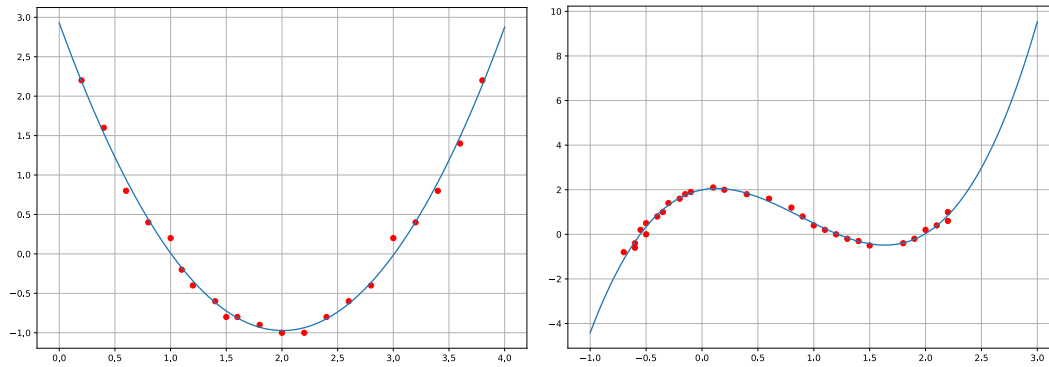
Подставляя в формулу из метода Ньютона получаем

$$x^{k+1} = x^k - (J_r(x^k)^T J_r(x^k))^{-1} \cdot J_r(x^k)^T r(x^k)$$

По хорошему еще стоит добавить коэффициент обучения

$$x^{k+1} = x^k - \alpha \cdot (J_r(x^k)^T J_r(x^k))^{-1} \cdot J_r(x^k)^T r(x^k)$$





Quadratic and Cubic Regressions

1.2 Powell Dog Leg

Для начала давайте поговорим о том, что такое Trust Region. Доверительная область - это область вокруг текущей точки, в которой мы считаем, что находится минимум функции. В этой области осуществляется поиск новых точек, которые могут быть более оптимальные чем текущая. Она задается радиусом и центром (для начала радиус берется очень большой, чтобы покрыть большую область поскольку таким образом увеличивается вероятность, что мы найдем оптимальные для нас стартовые точки).

Как она изменяется? В процессе работы алгоритма, если находится новая оптимальная точка, то доверительная область (этот круг) сдвигается к ней, если нет, то уменьшается радиус круга и снова ищется оптимальная точка.

Проблемы такой стратегии: заикливание. Алгоритм может заиклиться при уменьшении радиуса круга, когда новой оптимальной точки не появляется. Эта проблема решается установлением нижнего предела для радиуса, при котором алгоритм закончит работу. Trust-region метод использует квадратичную модель. На каждой итерации шаг вычисляется путем решения следующей квадратичной задачи:

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} \cdot p^T H p$$

Хотим делать шаг с одной стороны находя минимум этой модели, но с другой стороны, не выходить за пределы нашего доверительного региона, так как за его пределами приближение может работать довольно-таки плохо. Таким образом минимизировать $\min_{\|p\| \leq \Delta_k} m(p)$, где Δ_k - радиус доверительного региона. p^U - минимум вдоль направления градиента, p^H - минимум квадратичной модели.

$$p^U = -\frac{\nabla f^T \nabla f}{\nabla f^T H \nabla f}$$

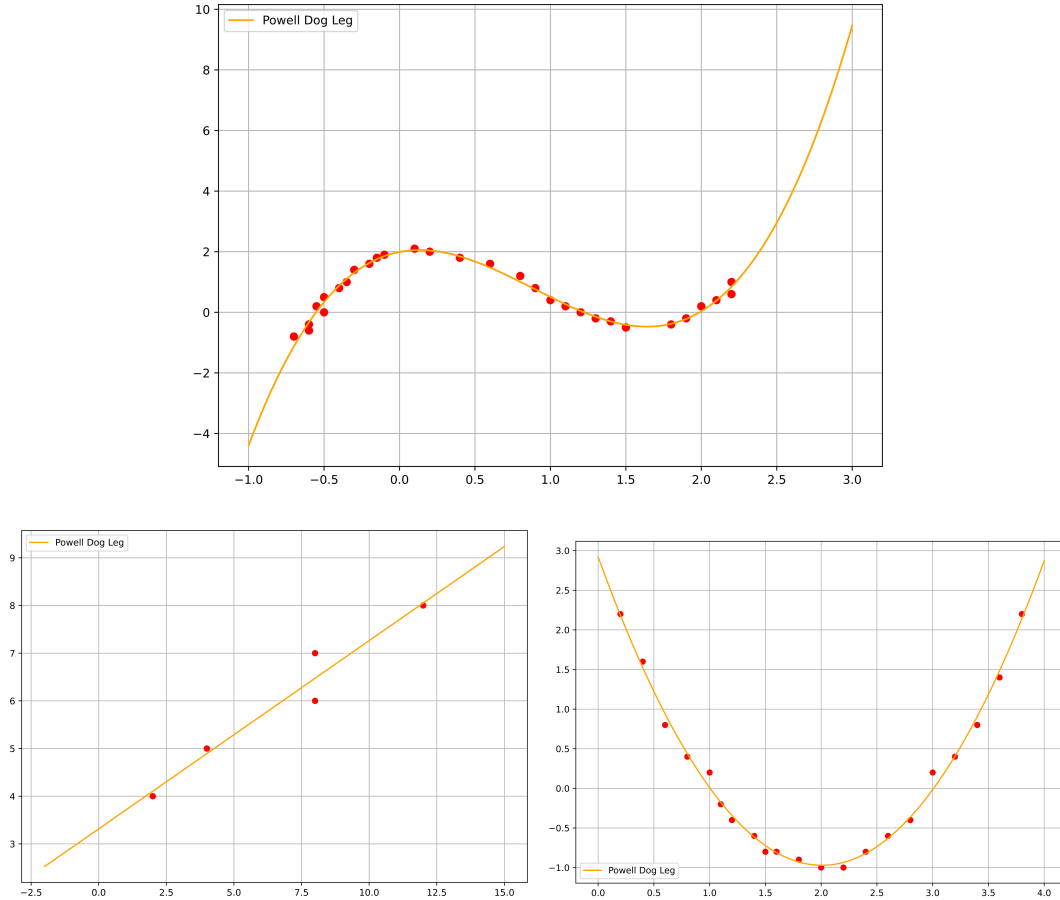
$$p^H = -\frac{\nabla f}{H}$$

1. Если вся собачья нога находится внутри доверительного региона, то мы шагаем туда. Если нет, то хотим найти пересечение ломанной с нашей сферой.
2. Если p^U находится на расстоянии больше чем наш радиус, то находим их пересечение. Если меньше, то пересекаем отрезок между p^U и p^H со сферой.

Первая проблема, которая возникает при определении trust-region алгоритма — это выбор стратегии для поиска оптимального trust-region радиуса на каждой итерации. Выбор основывается на сходстве функции m_k и целевой функции f на предыдущей итерации. Далее мы определяем следующее соотношение:

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$$

Если ρ_k отрицательное или близкое к 0, мы уменьшаем размер trust-region области, так как модель плохо отражает целевую функцию. Если ρ_k большое, тогда модель хорошо соответствует целевой функции. В данном случае следует расширить trust-region на следующей итерации.



2 Broyden Fletcher Goldfarb Shanno

2.1 Theory

BFGS - Квази-Ньютоновский метод, это значит, что нам не нужен Гессиан, который, как мы знаем достаточно дорого считается. Мы введем матрицу B , которую мы будем обновлять на каждом шаге, она будет служить нашим приближением к гессиану H

$$x^{k+1} = x^k - \alpha_k B_k^{-1} \nabla f(x^k)$$

Понятно, что если мы вместо B подставим гессиан, мы получим всем известный метод Ньютона, наш алгоритм будет основан на том, что нам не надо тратить много вычислений на Гессиан, мы не будем сходиться квадратично, как в методе Ньютона, но и не будем сходиться линейно, как в обычном градиентном спуске.

В методе BFGS используется данное приближение

$$B_{k+1}(x^{k+1} - x^k) = \nabla f(x^{k+1}) - \nabla f(x^k)$$

К сожалению при $n > 1$ у нас на руках неопределенная система, в том плане, что $\frac{n(n+1)}{2}$ переменных в B , а всего n уравнений. Чтобы разрешить нашу проблему BFGS предлагает нам добавить несколько ограничений.

$$\min ||B_{k+1} - B_k||$$

$$B_{k+1}^T = B_{k+1}$$

$$B_{k+1}\Delta x_k = y_k, y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$$

Эти ограничения в совокупности обеспечивают, что полученная матрица B будет симметричной и положительно определенной, и что B_{k+1} будет не сильно отличаться от B_k . Итого, мы получаем

$$B_{k+1} = B_k + \alpha uu^T + \beta vv^T$$

где α и β - скаляры, а u и v - векторы. Подставим $u = y_k$ и $v = B_k s_k$ где $s_k = x^{k+1} - x^k$

$$B_{k+1} = B_k + \alpha y_k y_k^T + \beta B_k s_k s_k^T B_k$$

$$y_k = B_{k+1} s_k = B_k s_k + \alpha y_k y_k^T s_k + \beta B_k s_k s_k^T B_k s_k$$

Домножив на s_k мы получили y_k . Мы теперь можем подогнать α и β такими, чтобы выходило

ровно y_k . Чтобы все сокращалось подставим $\alpha = \frac{1}{y_k^T s_k}$ и $\beta = -\frac{1}{s_k^T B_k s_k}$

$$y_k = B_k s_k + \frac{1}{y_k^T s_k} y_k y_k^T s_k + -\frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k s_k$$

$$y_k = y_k$$

Итого, подставив данные о α и β в формулу приближения Гессиана получаем

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

Получается на каждой итерации нашего алгоритма мы сперва ищем шаг s_k

$$B_k s_k = -\nabla f(x^k)$$

Двигаемся на шаг

$$x^{k+1} = x^k + s_k$$

Вычисляем изменение значения функции y_k

$$y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$$

И наконец, получив s_k , y_k обновляем приближение к Гесссиану

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

Можно заметить, что мы упускаем, что-то важное... Квази-Ньютоновские алгоритмы имеют вид

$$x^{k+1} = x^k - \alpha_k B_k^{-1} \nabla f(x^k)$$

Получается на каждой итерации нам нужно находить обратную к B , это куб операций на каждой итерации, поэтому давайте, оптимизируем и вместо того, чтобы на каждом шаге вычислять B_k будем вычислять B_k^{-1}

Выберем какое-то начальное положение x_0 и начальное приближение к Гессе, пусть это будет единичная матрица $H = I$, на каждой итерации

$$s_k = -H_k \nabla f(x^k)$$

Обновляем положение

$$x^{k+1} = x^k + s_k$$

Вычисляем изменение значения функции y_k

$$y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$$

Обновляем приближение к матрице обратной Гесссиану

$$H_{k+1} = (I - s_k \rho_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k}$$

3 L-BFGS

L в LBFGS означает - limited memory, то есть ограничение по памяти. Вместо того, чтобы, как в обычном BFGS хранить Гесссиан, как матрицу n на n будем хранить несколько векторов, которые нам будут говорить о его виде.

Мы будем хранить пары состоящие из y_k и s_k , запись из предыдущих пунктов к $x^{k+1} - x^k$ и $\nabla f(x^{k+1}) - \nabla f(x^k)$. Что не менее важно, мы будем хранить последние m штук, это нам поможет и с памятью, и, возможно, со сходимостью, ибо у нас данные о Гесссиане будут более актуальными.

$$\begin{aligned} H_{k+1} &= (I - s_k \rho_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k} \\ H_k &= (I - s_{k-1} \rho_{k-1} y_{k-1}^T) H_{k-1} (I - \rho_{k-1} y_{k-1} s_{k-1}^T) + \rho_{k-1} s_{k-1} s_{k-1}^T \\ &\vdots \\ H_1 &= (I - s_0 \rho_0 y_0^T) H_0 (I - \rho_0 y_0 s_0^T) + \rho_0 s_0 s_0^T, \quad H_0 = I \end{aligned}$$

В итоге получается, что затраты на память у нас буду линейные, и наиболее актуальный Гесссиан, что является причиной широкой распространенности данного метода.

4 Compare and Contrast

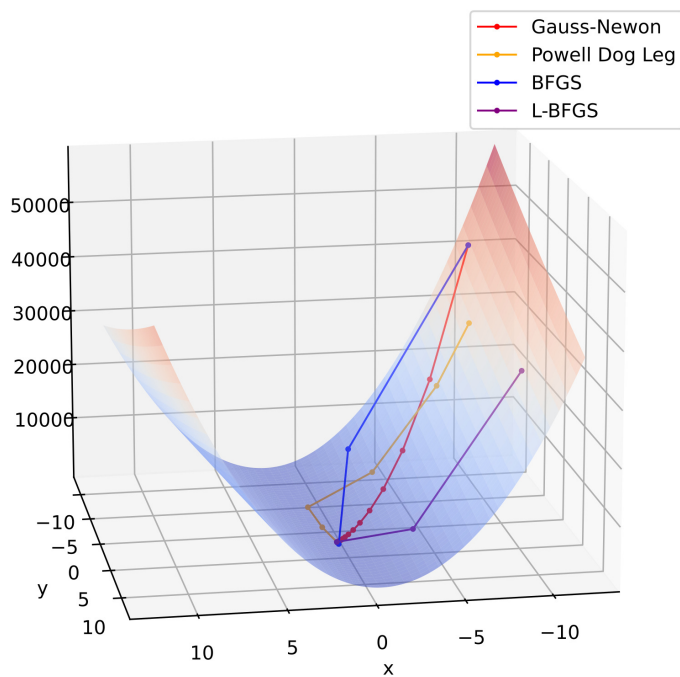
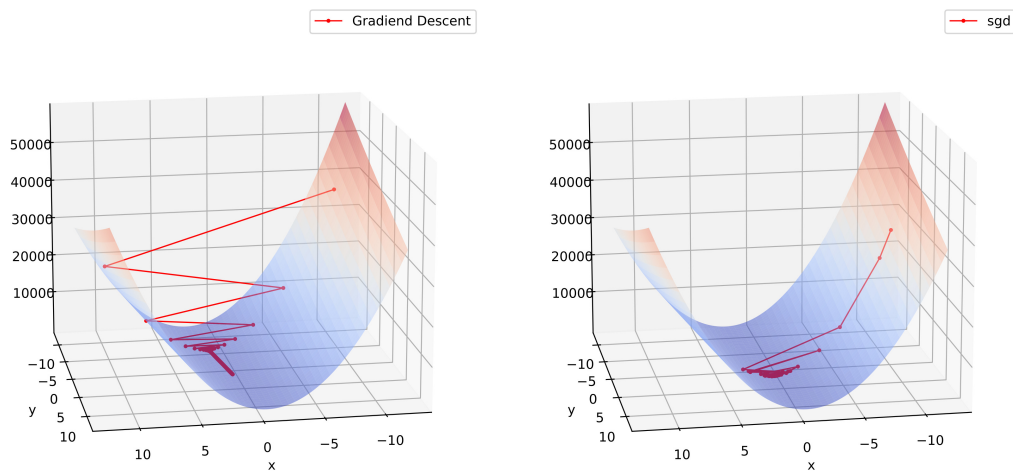
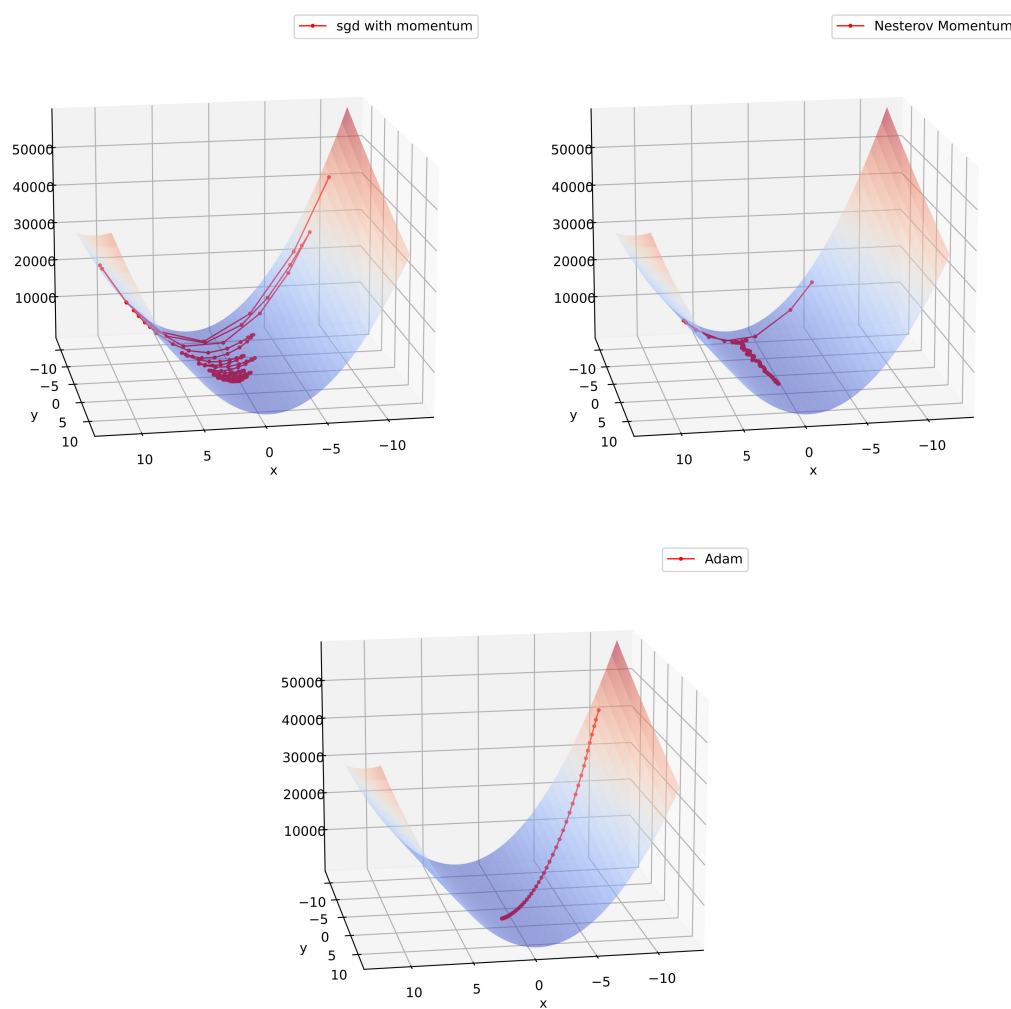


Рис. 1: All Methods side-by-side

Посмотрим на работу методов из предыдущих работ, то есть градиентный спуск, сгд, моментум, адам итп.





На графиках мы явно наблюдаем преимущество Квази-Ньютоновских алгоритмов, но это и не удивительно, мы смотрим на приближение гессиана, что позволяет нам понять на сколько мы можем шагнуть, поэтому сходимость не линейная, а суперлинейная. В методе Ньютона, у нас квадратичная сходимость, но мы на точное вычисление Гессиана будем тратить много ресурсов.

Gauss-Newton	PDL	BFGS	L- BFGS
(0.393342, 3.314003)	(0.394736, 3.315789)	(0.394736, 3.315789)	(0.394498, 3.317877)

Таблица 1: Convergence 1

Gradiend Descent	SGD	Adam	Nesterov
(0.394571, 3.317834)	(0.510510, 2.851005)	(0.341868, 3.321072)	(0.344294, 3.592078)

Таблица 2: Convergence 2

Gauss-Newton	PDL	BFGS	L-BFGS
1036562900	1251134567	2205300	2892300

Таблица 3: Time (ns) 1

Gradiend Descent	SGD	Adam	Nesterov
932050900	883746000	933071400	1020918400

Таблица 4: Time (ns) 2

Gauss-Newton	PDL	BFGS	L-BFGS
130	132	291	156

Таблица 5: Memory (MiB) 1

Gradiend Descent	SGD	Adam	Nesterov
270.2	261.36	264.62	262.61

Таблица 6: Memory (MiB) 2

5 Conclusion

Были исследованы методы Гаусса-Ньютона, Powell Dog Leg, BFGS и L-BFGS для нелинейной регрессии. Также мы сравнили эффективность этих алгоритмов между собой и с алгоритмами из предыдущих работ.